iris_localization.h File Reference

individuazione dell'iride e della pupilla More...

```
#include <opencv2/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <math.h>
#include <cmath>
#include <algorithm>
#include <vector>
#include <assert.h>
#include <numeric>
#include <tuple>
#include "reflection_correction.h"
#include "preprocessing.h"
#include "normalization.h"
#include "test.h"
#include "occlusion_reflection_detec.h"
#include <chrono>
#include <ctime>
#include <dirent.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Go to the source code of this file.

Classes

struct	**image**
struct	**results**

Macros

#define	**DELTA_R** 5

Functions

uchar **pixel_value** (**image** *img, double angle, Point centro, int r)
calcolo il valore del pixel preso in considerazione (x,y) secondo la modalità LUA (somma pesata per percezione) More...

uchar **pixel_value_multi** (**image** *img, double angle, Point centro, int r, string col)
ritorno il valore del pixel preso in considerazione (x,y) per lo spettro da considerare (col) More...

int **contour_sum** (**image** *img, Point centro, int r)
calcolo la somma dei valori presenti sui punti della circonferenza More...

int	**contour_sum_multi** (**image** *img, Point centro, int r, string col) calcolo la somma dei valori presenti sui punti della circonferenza More...
vector< int >	**linear_integral_vector** (**image** *img, Point centro, vector< int > radius_range) calcolo l'integrale di linea dei valori con posizione sulla circonferenza del cerchio di raggio r, con r appartenente a radius_range More...
vector< int >	**linear_integral_vector_multi** (**image** *img, Point centro, vector< int > radius_range, string col) integrale di linea dei valori con posizione sulla circonferenza del cerchio di raggio r, con r appartenente a radius_range. Modalità multi-spettro, per cui calcolo l'integrale su singolo spettro. More...
pair< int, int >	**obtain_w_h** (int cols, int rows) calcola valori di width e height in modo da utilizzarli per ridimensionare le immagini e rendere la computazione dell'algoritmo quanto più veloce possibile More...
pair< int, int >	**obtain_w_h_miche** (int cols, int rows) calcolo i valori di width e height da utilizzare per ridimensionare l'immagine (appartenente a MICHE) More...
int	**checkWidth** (int width) ricerca di k, valore che verrà successivamente valutato per capire se è possibile o meno ridimensionare l'immagine con un coefficiente che non modifichi di height e width rendendo l'immagine rumorosa More...
results *	**apply_daugman_operator** (**image** *img, int r_min, int r_max) creo le directory presenti in inp_path all'interno di out_path e ritorno i rispettivi path (ritorno out_vec che contiene tali path). N.B.: valido per Utiris More...
results *	**apply_daugman_operator_multi** (**image** *img, int r_min, int r_max) applicazione del multi-spectral integro-differential operator di Daugman (vedi Krupicka e Daugman) in modalità multi-spettro discretizzato per ottenere il contorno dell'iride (limbus) More...
results *	**pupil_daugman_operator** (Mat *img_red, int r_min, int r_max) applicazione dell'integro-differential operator di Daugman (non modificato, vedi Daugman) discretizzato per ottenere il contorno della pupilla More...
vector< int >	**pupil_linear_integral_vector** (Mat *img_red, Point centro, vector< int > radius_range) integrale di linea dei valori con posizione sulla circonferenza del cerchio di raggio r, con r appartenente a radius_range More...
int	**pupil_contour_sum** (Mat *img_red, Point centro, int r) calcolo la somma dei valori dei pixel sulla circonferenza di con punto centrale centro e raggio r More...
uchar	**pupil_pixel_value** (Mat *img_red, double angle, Point centro, int r) calcolo le coordinate del pixel (x,y) usando angolo, centro e raggio dela circonferenza per ottenere il valore del pixel More...
int	**pup_contour_divider** (Mat *img_red, vector< int > radius_range, int pos, Point centro) pup_contour_divider ritorna la somma dei pixel sul perimetro della circonferenza con centro(x,y) "centro" e raggio radius_range[pos-2] dove pos è la posizione del kernel sul vettore degli integrali di linea. Per maggiori dettagli, vedi pup_convolution e l'operatore di

	Daugman discretizzato, in particolare l'operatore discretizzato modificato per il calcolo della pupilla More...
Mat	**pup_convolution** (Mat *img_red, double sigma, vector< int > *line_int, vector< int > radius_range, Point centro) convoluzione tra kernel gaussiano (di dimensione DELTA_R e coefficiente sigma) e vettore dell'integrale lineare N.B.: ogni elemento corrisponde ad un integrale di linea con un determinato raggio, il punto centrale preso in considerazione è lo stesso per tutti i valori del vettore inoltre, a differenza di convolution, divido ogni elemento uscente dal calcolo della convoluzione con il valore (somma) dei pixel sul perimetro della ciconferenza con raggio r-2 dove r è il raggio preso in considerazione come punto centrale della convoluzione More...
Mat	**convolution** (double sigma, vector< int > *line_int) convoluzione tra kernel gaussiano (di dimensione DELTA_R e coefficiente sigma) e vettore dell'integrale lineare N.B.: ogni elemento corrisponde ad un integrale di linea con un determinato raggio, il punto centrale preso in considerazione è lo stesso per tutti i valori del vettore More...
double	**pixel_conv** (vector< int > *line_int, vector< double > *kernel, int pos) con pixel_conv, ed in particolare con pixel, qui si intende il singolo elemento del vettore line_int su cui viene fatta convoluzione con il kernel gaussiano More...
Mat	**binaryMask** (Mat *src_image, int iris_r, Point iris_c, int pupil_r, Point pupil_c) viene creata una maschera binaria (coordinate cartesiane) per l'iride e pupilla individuate sull'immagine di input dell'algoritmo More...
vector< string >	**split** (string strToSplit, char delimeter) split per string More...

Detailed Description

individuazione dell'iride e della pupilla

Function Documentation

◆ apply_daugman_operator()

```
results* apply_daugman_operator ( image *  img,
                                  int       r_min,
                                  int       r_max
                                )
```

creo le directory presenti in inp_path all'interno di out_path e ritorno i rispettivi path (ritorno out_vec che contiene tali path). N.B.: valido per Utiris

Parameters

inp_path path alla directory di input (in cui si trova il DB, in questo caso Utiris)

out_path path alla directory di output in cui andranno a crearsi tante dirs quante sono in inp_path

Returns

vector<string> out_vec contenente i path alle singole directory create applicazione dell'operatore integro-differenziale di Daugman (vedi Krupicka e Daugman) in modalità LUA discretizzato per ottenere il contorno dell'iride (limbus)

Parameters

img struct contenente matrici dei 3 canali (BGR), numero di colonne, numero di righe (width e height)

r_min raggio minimo preso in considerazione. Di norma equivale a height/4

r_max raggio massimo preso in considerazione. Di norma equivale a height/2

Returns

results struct contenente raggio, punto centrale (x,y) e valore massimo. Il valore massimo equivale al valore massimo che l'operatore integro-differenziale ottiene in base ai vari raggi e punti centrali presi in considerazione

◆ apply_daugman_operator_multi()

```
results* apply_daugman_operator_multi ( image *  img,
                                        int      r_min,
                                        int      r_max
                                      )
```

applicazione del multi-spectral integro-differential operator di Daugman (vedi Krupicka e Daugman) in modalità multi-spettro discretizzato per ottenere il contorno dell'iride (limbus)

Parameters

img	struct contenente matrici dei 3 canali (BGR), numero di colonne, numero di righe (width e height)
r_min	raggio minimo preso in considerazione. Di norma equivale a height/4
r_max	raggio massimo preso in considerazione. Di norma equivale a height/2

Returns

results struct contenente raggio, punto centrale (x,y) e valore massimo. Il valore massimo equivale al valore massimo che l'operatore integro-differenziale ottiene in base ai vari raggi e punti centrali presi in considerazione

◆ binaryMask()

```
Mat binaryMask ( Mat *  src_image,
                 int    iris_r,
                 Point  iris_c,
                 int    pupil_r,
                 Point  pupil_c
               )
```

viene creata una maschera binaria (coordinate cartesiane) per l'iride e pupilla individuate sull'immagine di input dell'algoritmo

Parameters

src_image	immagine da utilizzare per creare la maschera binaria
iris_r	raggio dell'iride individuata
iris_c	centro dell'iride individuata
pupil_r	raggio della pupilla individuata
pupil_c	centro della pupilla individuata

Returns

inizializzazione della maschera binaria dell'immagine originale (non normalizzata). Inizializzata valutando tutti i pixel tra iride e pupilla.

◆ checkWidth()

```
int checkWidth ( int  width )
```

ricerca di k, valore che verrà successivamente valutato per capire se è possibile o meno ridimensionare l'immagine con un coefficiente che non modifichi di height e width rendendo l'immagine rumorosa

Parameters

width numero di colonne (larghezza) dell'immagine

Returns

intero k utilizzato per valutare se allargare o diminuire la lunghezza dell'immagine, in modo da avere un coefficiente con cui dividere equamente width e height senza stravolgere l'immagine

◆ contour_sum()

```
int contour_sum ( image *  img,
                  Point    centro,
                  int      r
                )
```

calcolo la somma dei valori presenti sui punti della circonferenza

Parameters

img puntatore alla struct image (contenente puntatori alle matrici dei tre spettri di colore dell'immagine)

centro punto centrale (x,y) della circonferenza presa in considerazione

r raggio della circonferenza presa in considerazione

Returns

somma di valori del perimetro della circonferenza presa in considerazione n.b.: questa somma rappresenta l'integrale lineare chiuso secondo la formula di Daugman discretizzata

◆ contour_sum_multi()

```
int contour_sum_multi ( image *  img,
                        Point    centro,
                        int      r,
                        string   col
                      )
```

calcolo la somma dei valori presenti sui punti della circonferenza

Parameters

- **img** puntatore alla struct image (contenente puntatori alle matrici dei tre spettri di colore dell'immagine)
- **centro** punto centrale (x,y) della circonferenza presa in considerazione
- **r** raggio della circonferenza presa in considerazione
- **col** colore dello spettro da utilizzare nell'immagine img

Returns

somma di valori del perimetro della circonferenza presa in considerazione n.b.: questa somma rappresenta l'integrale lineare chiuso secondo la formula di Daugman discretizzata

◆ convolution()

```
Mat convolution ( double           sigma,
                  vector< int > *  line_int
                )
```

convoluzione tra kernel gaussiano (di dimensione DELTA_R e coefficiente sigma) e vettore dell'integrale lineare N.B.: ogni elemento corrisponde ad un integrale di linea con un determinato raggio, il punto centrale preso in considerazione è lo stesso per tutti i valori del vettore

Parameters

- **sigma** coefficiente per il calcolo del gaussian kernel
- **line_int** vettore contenente l'integrale lineare per tutti i raggi (r_min fino a r_max, con indice crescente corrisponde un raggio maggiore del precedente) per un determinato punto centrale

Returns

matrice a due righe ed n colonne, ogni colonna corrisponde a un valore di convoluzione dell'integrale lineare con kernel gaussiano, convoluzione applicata con punto centrale del kernel sul valore in posizione pos del vettore dell'integrale lineare. La prima riga è per n-k (vedi Daugman discretizzato), la seconda riga è per n-k-1

◆ linear_integral_vector()

```
vector<int> linear_integral_vector ( image *        img,
                                     Point          centro,
                                     vector< int >  radius_range
                                   )
```

calcolo l'integrale di linea dei valori con posizione sulla circonferenza del cerchio di raggio r, con r appartenente a radius_range

Parameters

img	puntatore alla struct image (contenente puntatori alle matrici dei tre spettri di colore dell'immagine)
centro	punto centrale (x,y) della circonferenza presa in considerazione
radius_range	vettore contenente i raggi presi in considerazione (da r_min a r_max)

Returns

line_integral vettore contenente un integrale di linea per ogni raggio presente in radius_range, integrale di linea sulla circonferenza con origine pari a centro

◆ linear_integral_vector_multi()

```
vector<int> linear_integral_vector_multi ( image *        img,
                                           Point          centro,
                                           vector< int >  radius_range,
                                           string         col
                                         )
```

integrale di linea dei valori con posizione sulla circonferenza del cerchio di raggio r, con r appartenente a radius_range. Modalità multi-spettro, per cui calcolo l'integrale su singolo spettro.

Parameters

img	puntatore alla struct image (contenente puntatori alle matrici dei tre spettri di colore dell'immagine)
centro	punto centrale (x,y) della circonferenza presa in considerazione
radius_range	vettore contenente i raggi presi in considerazione (da r_min a r_max)
col	colore ("red", "blue", "green") dello spettro utilizzato

Returns

line_integral vettore contenente un integrale di linea per ogni raggio presente in radius_range, integrale di linea sulla circonferenza con origine pari a centro

◆ obtain_w_h()

```
pair<int,int> obtain_w_h ( int  cols,
                           int  rows
                         )
```

calcola valori di width e height in modo da utilizzarli per ridimensionare le immagini e rendere la computazione dell'algoritmo quanto più veloce possibile

Parameters

cols colonne (width) della matrice (immagine) presa in considerazione

rows righe (height) della matrice (immagine) presa in considerazione

Returns

pair<int,int> contenente cols/k e rows/k, k è scelto in base al valore che più si avvicina al quoziente colonne/256 in modo da poter ridimensionare tutte le immagini allo stesso modo e con width <= 256 (stesso discorso vale per l'height, ammesso che cols>=rows)

◆ obtain_w_h_miche()

```
pair<int,int> obtain_w_h_miche ( int  cols,
                                 int  rows
                               )
```

calcolo i valori di width e height da utilizzare per ridimensionare l'immagine (appartenente a MICHE)

Parameters

cols colonne (width) della matrice (immagine) presa in considerazione

rows righe (height) della matrice (immagine) presa in considerazione

Returns

pair<int,int> contenente cols/k e rows/k

◆ pixel_conv()

```
double pixel_conv ( vector< int > *        line_int,
                    vector< double > *     kernel,
                    int                    pos
                  )
```

con pixel_conv, ed in particolare con pixel, qui si intende il singolo elemento del vettore line_int su cui viene fatta convoluzione con il kernel gaussiano

Parameters

line_in	vettore contenente l'integrale lineare per tutti i raggi (r_min fino a r_max, con indice crescente corrisponde un raggio maggiore del precedente) per un determinato punto centrale
kernel	kernel gaussiano da applicare al vettore line_int
pos	posizione dell'elemento in line_int su cui l'elemento centrale del kernel si sovrappone

Returns

valore ottenuto dalla convoluzione tra il kernel gaussiano e gli elementi del vettore degli integrali sul quale il kernel si sovrappone

◆ pixel_value()

```
uchar pixel_value ( image *  img,
                    double   angle,
                    Point    centro,
                    int      r
                  )
```

calcolo il valore del pixel preso in considerazione (x,y) secondo la modalità LUA (somma pesata per percezione)

Parameters

img	puntatore alla struct image (contenente puntatori alle matrici dei tre spettri di colore dell'immagine)
angle	attuale angolo della circonferenza (presa in considerazione)
centro	punto centrale (x,y) della circonferenza presa in considerazione
r	raggio della circonferenza presa in considerazione

Returns

somma pesata del pixel sui tre canali BGR, secondo la formula B[pixel]*0.114 + G[pixel]*0.587 + R[pixel]*0.299

◆ pixel_value_multi()

```
uchar pixel_value_multi ( image *  img,
                          double   angle,
                          Point    centro,
                          int      r,
                          string   col
                        )
```

ritorno il valore del pixel preso in considerazione (x,y) per lo spettro da considerare (col)

Parameters

img	puntatore alla struct image (contenente puntatori alle matrici dei tre spettri di colore dell'immagine)
angle	attuale angolo della circonferenza (presa in considerazione)
centro	punto centrale (x,y) della circonferenza presa in considerazione
r	raggio della circonferenza presa in considerazione
col	spettro dell'immagine da prendere in considerazione ("blue", "green", "red")

Returns

valore del pixel (x,y) dello spettro col

◆ pup_contour_divider()

```
int pup_contour_divider ( Mat *          img_red,
                          vector< int >  radius_range,
                          int            pos,
                          Point          centro
                        )
```

pup_contour_divider ritorna la somma dei pixel sul perimetro della circonferenza con centro(x,y) "centro" e raggio radius_range[pos-2] dove pos è la posizione del kernel sul vettore degli integrali di linea. Per maggiori dettagli, vedi pup_convolution e l'operatore di Daugman discretizzato, in particolare l'operatore discretizzato modificato per il calcolo della pupilla

Parameters

img_red	matrice del canalre rosso (BGR) dell'immagine presa in considerazione
radius_range	vettore contenente i raggi presi in considerazione (da r_min a r_max)
pos	valore centrale del kernel della convoluzione. N.B.: questa pup_contour_divider viene chiamato SOLO da pup_convolution, che gli da' in input pos. pos è necessario per ottenere il raggio r-2
centro	punto centrale (x,y) della circonferenza presa in considerazione

Returns

ritorna la somma dei pixel presenti sul perimetro della circonferenza presa in considerazione

◆ pup_convolution()

```
Mat pup_convolution ( Mat *          img_red,
                      double         sigma,
                      vector< int > * line_int,
                      vector< int >  radius_range,
                      Point          centro
                    )
```

convoluzione tra kernel gaussiano (di dimensione DELTA_R e coefficiente sigma) e vettore dell'integrale lineare N.B.: ogni elemento corrisponde ad un integrale di linea con un determinato raggio, il punto centrale preso in considerazione è lo stesso per tutti i valori del vettore inoltre, a differenza di convolution, divido ogni elemento uscente dal calcolo della convoluzione con il valore (somma) dei pixel sul perimetro della ciconferenza con raggio r-2 dove r è il raggio preso in considerazione come punto centrale della convoluzione

Parameters

img_red	matrice del canalre rosso (BGR) dell'immagine presa in considerazione
sigma	coefficiente per il calcolo del gaussian kernel
line_in	vettore contenente l'integrale lineare per tutti i raggi (r_min fino a r_max, con indice crescente corrisponde un raggio maggiore del precedente) per un determinato punto centrale
radius_range	vettore contenente i raggi presi in considerazione (da r_min a r_max)

Returns

matrice a due righe ed n colonne, ogni colonna corrisponde a un valore di convoluzione dell'integrale lineare con kernel gaussiano, convoluzione applicata con punto centrale del kernel sul valore in posizione pos del vettore dell'integrale lineare. La prima riga è per n-k (vedi Daugman discretizzato), la seconda riga è per n-k-1

◆ pupil_contour_sum()

```
int pupil_contour_sum ( Mat *  img_red,
                        Point  centro,
                        int    r
                      )
```

calcolo la somma dei valori dei pixel sulla circonferenza di con punto centrale centro e raggio r

See also

contour_sum la differenza è che per la pupilla calcolo l'integrale discretizzato (somma dei pixel sul perimetro della circonferenza) per l'intera circonferenza, perché si suppone non ci siano occlusioni dovute a ciglia e/o palpebra superiore

Parameters

img_red	matrice del canale RED dell'immagine di input
centro	punto centrale (x,y) della circonferenza presa in considerazione
r	raggio della circonferenza presa in considerazione

Returns

somma dei valori dei pixel sulla circonferenza

◆ pupil_daugman_operator()

```
results* pupil_daugman_operator ( Mat *  img_red,
                                  int    r_min,
                                  int    r_max
                                )
```

applicazione dell'integro-differential operator di Daugman (non modificato, vedi Daugman) discretizzato per ottenere il contorno della pupilla

Parameters

img_red	matrice del canale RED dell'immagine di input
r_min	raggio minimo preso in considerazione. Per la pupilla è pari al raggio dell'iride ottenuta diviso per 5
r_max	raggio massimo preso in considerazione. Per la pupilla è pari al raggio dell'iride ottenuta moltiplicato per 0.75

Returns

results struct contenente raggio, punto centrale (x,y) e valore massimo. Il valore massimo equivale al valore massimo che l'operatore integro-differenziale ottiene in base ai vari raggi e punti centrali presi in considerazione

◆ pupil_linear_integral_vector()

```
vector<int> pupil_linear_integral_vector ( Mat *          img_red,
                                           Point          centro,
                                           vector< int >  radius_range
                                         )
```

integrale di linea dei valori con posizione sulla circonferenza del cerchio di raggio r, con r appartenente a radius_range

See also

linear_integral_vector la differenza è che per la pupilla tengo in considerazione, e calcolo l'integrale, per l'intera circonferenza

Parameters

img_red	matrice del canale RED dell'immagine di input. Utilizzo solo lo spettro di colore rosso perché permette una migliore visione della pupilla
centro	punto centrale (x,y) della circonferenza presa in considerazione
radius_range	vettore contenente i raggi presi in considerazione (da r_min a r_max)

Returns

line_integral vettore contenente un integrale di linea per ogni raggio presente in radius_range, integrale di linea sulla circonferenza con origine pari a centro

◆ pupil_pixel_value()

```
uchar pupil_pixel_value ( Mat *  img_red,
                          double angle,
                          Point  centro,
                          int    r
                        )
```

calcolo le coordinate del pixel (x,y) usando angolo, centro e raggio dela circonferenza per ottenere il valore del pixel

See also

pixel_value la differenza è che per la pupilla, avendo un singolo canale (il rosso) non faccio nessuna somma pesata

Parameters

img_red	matrice del canale RED dell'immagine di input
angle	angolo attualmente preso in considerazione, viene utilizzato nel calcolo delle coordinate del punto (x,y) all'angolo angle della circonferenza presa in considerazione
centro	punto centrale (x,y) della circonferenza presa in considerazione
r	raggio della circonferenza presa in considerazione

Returns

valore del pixel (x,y) sulla circonferenza della pupilla con raggio r

◆ split()

```
vector<string> split ( string  strToSplit,
                       char    delimeter
                     )
```

split per string

Parameters

strToSplit	stringa sul quale effettuare lo split
delimiter	delimitatore usato per effettuare lo split

Returns

vector<string> contenente gli elementi della stringa al quale è stato effettuato lo split